Pseudocode for Bayesian Network

Linjian Li

Begin at October, 2018

In this document, the term "variable" and "node" are used interchangeably.

1 Data Structures

```
Trainer {
    trainingSet: a m-by-n matrix where m is the number of training samples and n is the number of the feature variables of each sample
    incompleteData: a list containing the variables' name of the missing data in the training set
    incompleteDataExpectation: a **map** of lists of **pair**s, each pair contain the partial data case and the corresponding probabilities (weights), the map's keys are the variables' name
}

Network {
    nodesContainer: a list storing all nodes in this network
    edgesContainer: a list storing all edges in this network
}

Node {
    parents: a list storing this node's parents nodes' names
    children: a list storing this node's children nodes' names
    potentialValues: a list storing this node's potential values, which represents the row heading of the conditional probability table
    parentsCombinations: a list storing combinations of potential values of parents nodes, which represents the column heading of the conditional probability table. the elements of list are **set**s and the elements of these sets are **pair**s of variable's name and corresponding value
    condProbTable: a matrix, which is a **map** of **map**s, representing conditional probability table of this node. for outer map, the keys of are this node's potential values. for the inner maps the keys are combinations of parents nodes.
    margProbTable: a list storing the marginal probability of potential values of this node
}

Conditional Probability Table of A Node {
    An m-by-n matrix, where m is the number of potential values of this node and n is the number of the combinations of potential values of parents nodes.That is, CPT will have one row for each potential value of this node, and will have one column for each possible combination of values of the parents nodes. Each column must sum up to 1, but each row may not.
    The implementation of this matrix will use **map** of **map**s. For example, suppose a node, Z, has two parents, X and Y, and the corresponding potential values are $Z \in \{t, f\}$, $X \in \{1, 2\}$ and $Y \in \{a, b\}$. So, the element at Z.CPT[t][{(X,2),(Y,a)}] will be the value of $P(Z = t|X = 2, Y = a)$.
}

Factor {
    relatedVariables: a **set** storing variables related to this factor
    combList: a list storing the combinations of this factor's related nodes' value, which represents the row heading of this factor. the elements of list are **set**s and the elements of these sets are **pair**s of variable's name and corresponding value
    potentialsList: a **map** whose keys are the elements in "combList" of this factor and values are the corresponding potentials
}
```

2 Chou-Liu Algorithm

Algorithm 1 Chou-Liu Algorithm

```
 1: n ← nodesContainer.size()
 2: MIT ← empty n-by-n matrix
 3: for i = 0 to n-1 do
 4:    for j = 0 to i-1 do
 5:       if i = j then
             MIT[i][j] ← -1    /* Mutual information between a node and itself must be high, but we don't want a node to
             link to itself, so we manually set the mutual information to be -1. */
 6:       else
 7:          $(X_i, X_j) \leftarrow (nodesContainer[i], nodesContainer[j])$
 8:          MIT[i][j] ← mutualInformation($X_i, X_j$)
 9:          MIT[j][i] ← MIT[i][j]    // MIT is symmetric.
10:       end if
11:    end for
12: end for
13: markSet ← empty set stroing indexes
14: i ← the index of node as the root chosen by hand
15: add i to markSet
16: // Prim's algorithm
17: while markSet.size() < n do
18:    (maxMutualInfo, maxI, maxJ) ← (-1, -1, -1)
19:    for i in markSet do
20:       for j = 0 to n-1 do
21:          if j not in markSet and MIT[i][j] > maxMutualInfo then
22:             (maxMutualInfo, maxI, maxJ) ← (MIT[i][j], i, j)
23:          end if
24:       end for
25:    end for
26:    add maxJ to markSet
27:    add edge $(maxI, maxJ)$ to edgesContainer
28: end while
29: topologicalSortedPermutation ← generate a list of indexes using width first traversal starting at nodesCon-
    tainer[markSet[0]]
30: for $e_{ij}$ in edgesContainer do
31:    $X_i \leftarrow nodesContainer[i]$
32:    $X_j \leftarrow nodesContainer[i]$
33:    if i comes before j in topologicalSortedPermutation then
34:       setParentChild($X_i, X_j$)
35:    else
36:       setParentChild($X_j, X_i$)
37:    end if
38: end for
```

3 Computing Mutual Information Between X_i and X_j with complete data

Algorithm 2 Computing Mutual Information Between X_i and X_j

```
m ← trainingSet.size()
$r_i$ ← $X_i$.potentialValues.size()
$r_j$ ← $X_j$.potentialValues.size()
$P_{ij}$ ← empty $r_i$-by-$r_j$ matrix
$P_i$ ← empty one-by-$r_i$ matrix
$P_j$ ← empty one-by-$r_j$ matrix
Initialize $P_{ij}$, $P_i$ and $P_j$ to be all zero
for a = 0 to $r_i$-1 do
    for b = 0 to $r_j$-1 do
        for s = 0 to m-1 do
            if trainingSet[s][i]=$X_i$.potentialValues[a] and trainingSet[s][j]=$X_j$.potentialValues[b] then
                $P_{ij}$[a][b] ← $P_{ij}$[a][b]+1
            end if
        end for
        $P_{ij}$[a][b] ← $P_{ij}$[a][b] / m
    end for
end for
for a = 0 to $r_i$-1 do
    for s = 0 to m-1 do
        if trainingSet[s][i]=$X_i$.potentialValues[a] then
            $P_i$[a] ← $P_i$[a]+1
        end if
    end for
    $P_i$[a] ← $P_i$[a] / m
end for
for b = 0 to $r_j$-1 do
    for s = 0 to m-1 do
        if trainingSet[s][j]=$X_j$.potentialValues[b] then
            $P_j$[b] ← $P_j$[b]+1
        end if
    end for
    $P_j$[b] ← $P_j$[b] / m
end for
mutualInformation ← 0
for a = 0 to $r_i$-1 do
    for b = 0 to $r_j$-1 do
        mutualInformation ← mutualInformation + $P_{ij}$[a][b] * $\log(\frac{P_{ij}[a][b]}{P_i[a]*P_j[b]})$
    end for
end for
return mutualInformation
```

4 Training with Known Structure and Complete and Incomplete Data

Algorithm 3 Training Bayesian Network

```
for i = 0 to nodesContainer.size()-1 do
  thisNode ← nodesContainer[i]
  CPT ← thisNode.condProbTable
  for comb in thisNode.parentsCombinations do
    denominator ← 0
    for s = 0 to trainingSet.size()-1 do
      compatibility ← calculate compatibility between comb and trainingSet[s]
      denominator ← denominator + compatibility
      query ← trainingSet[s][i]
      CPT[query][comb] ← CPT[query][comb] + compatibility
    end for
    for query in thisNode.potentialValues do
      CPT[query][comb] ← CPT[query][comb] / denominator
    end for
  end for
end for
```

What is the “compatibility” appearing in the algorithm above? My aim of introducing “compatibility” is to handle the cases where there are missing data. If there are missing data in a training sample, this sample will be split into several partial data cases with the corresponding probabilities (weights). So, if this training sample does not miss any data, the ”compatibility” will be zero or one depending on the corresponding values of the features. If this training sample does miss some data, then the “compatibility” is the probability (weight) of the corresponding partial data case.

5 Variable Elimination Algorithm

Finding elimination orderings.
https://www.coursera.org/lecture/probabilistic-graphical-models-2-inference/finding-elimination-ordering

- Greedy search using heuristic cost function
- Finding a low-width triangulation of the original graph

Possible cost functions:

- min-neighbours
- min-weight: weight (values) of factor formed
- min-fill: new fill edges (often used in practice)
- weighted min-fill: total weight of new fill edges (edge weight = product of weights of the 2 nodes)

Algorithm 4 Eliminate

```
1: Input:
   G: the graphical model,
   E: the observed evidences,
   Z: the set of variables to be eliminated with the elimination ordering ($Z_i$ comes before $Z_j$ iff $i < j$),
   Y: the set of query variables.
2: factors ← initialize(Z)
3: loadEvidence(factors, E)
4: F ← sumProductVariableElimination(factors, Z)
5: normalize(F)
```

6 Junction Tree Algorithm

Note that in function *Collect* and *Distribute*, when update the state of the cliques and separators, their scope (related variables) should not be changed, which means they should sum over the variables that do not belong to themselves.

7 Scoring Functions

Some scoring functions will yield the intermediate results whcih are too large for the computer. For example, the BDeu scoring function uses gamma function, and the input dataset may cause it to calculate the factorial of 2000. Therefore, we need to find another form of these scoring function to make them implementable.

7.1 K2

The original equation is in the paper *A Bayesian Method for the Induction of Probabilistic Networks from Data (Cooper, 1992)*, which is

$$P(B_S, D) = P(B_S) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!$$

If the dataset is large, then N_{ij} and N_{ijk} may be too large causing overflow. Maybe, changing the equation to the logarithm form is better.

$$log(P(B_S, D)) = log(P(B_S)) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} \left(log(\frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!}) + \sum_{k=1}^{r_i} log(N_{ijk}!) \right) \tag{1}$$

$$= log(P(B_S)) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} \left((\sum_{a=r_i-1}^{1} log(a)) - (\sum_{b=N_{ij}+r_i-1}^{1} log(b)) + \sum_{k=1}^{r_i} \sum_{c=N_{ijk}}^{1} log(c) \right) \tag{2}$$

7.2 BDe

The original equation is in *Learning Bayesian Networks The Combination of Knowledge and Statistical Data (Heckerman, 1995)*.

$$p(D, B_s^h | \xi) = p(B_s^h | \xi) \cdot \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})}$$

where $N'_{ijk} = N' p(x_i = k, \Pi_i = j | B_{sc}^h, \xi)$ and N' is the equivalent sample size.

After change it to the logarithm form, we can approximate the log of a Gamma function of a real number using the log of a factorial of an integer. That is,

$$log(\Gamma(x)) \approx log((ceiling(x) - 1)!)$$

Algorithm 5 Sub-processes Used in Elinimate

```
initialize(Z)
 1: factors ← empty list to store factors
 2: for i = 0 to size(Z)-1 do
 3:     factors[i] ← constructFactor(Z[i])
 4: end for
 5: return factors

loadEvidence(E)
 1: for i = 0 to factors.size()-1 do
 2:     if Z[i] is in E then
 3:         for each factor in factors related to variable Z[i] do
 4:             for comb in factor.combList do
 5:                 if comb is not compatible with E about Z[i] then
 6:                     factor.potentialsList[comb] ← 0      /* Instead of setting it to be zero, we can also simply drop it to save
                        space. But doing this may cause trouble when implementing using C++. How to deal with it? */
 7:                 end if
 8:             end for
 9:         end for
10:     end if
11: end for

sumProductVariableElimination(factors, Z)
 1: for i = 0 to Z.size()-1 do
 2:     tempList ← empty list to stroe factors
 3:     while factors contains elements related to Z[i] do
 4:         pop out one of these element in factors, and add it into tempList
 5:     end while
 6:     while tempList.size()¿1 do
 7:         tem1 ← pop out the first element in tempList
 8:         tem2 ← pop out the first element in tempList
 9:         product ← factorsProduct(temp1, temp2)
10:         add product into tempList
11:     end while
12:     product ← pop out the first element in tempList
13:     newFactor ← sumProductOverVariable(product, Z[i])
14:     add newFactor into factors
15: end for

normalize(F)
/* When run to this line, the only remain factor, F, will be the factor of variable Y, which is a CPT in the form of one-by-n
matrix, where n is the number of possible values of Y. */
 1: denominator ← 0
 2: for i = 0 to n-1 do
        denominator ← denominator + F[i]
 3: end for
 4: for i = 0 to n-1 do
        F[i] ← F[i] / denominator
 5: end for
```

Algorithm 6 Algorithms about Factors

```
constructFactor(node)
 1: nF ← new Factor
 2: nF.relatedVariables ← node.parents
 3: nF.relatedVariables.append(node)
 4: k ← 0
 5: for i in node.potentialValues do
 6:     for j in node.parentsCombinations do
 7:         comb = union(i, j)
 8:         nF.combList[k] ← comb
 9:         nF.potentialsList[comb] ← node.condProbTable[i][j]
10:         k ← k + 1
11:     end for
12: end for
13: return nF

factorsProduct(f1, f2)
 1: nF ← new Factor
 2: nF.relatedVariables ← union(f1.relatedVariables, f2.relatedVariables)
 3: k ← 0
 4: for i in f1.combList do
 5:     for j in f2.combList do
 6:         comb ← union(i,j)
 7:         nF.combList[k] ← comb
 8:         nF.potentialsList[comb] ← f1.potentialsList[i] * f2.potentialsList[j]
 9:         k ← k + 1
10:     end for
11: end for
12: return nF

sumProductOverVariable(f, v)
 1: temp ← f.potentialsList[i]
 2: nF ← new Factor
 3: nF.relatedVariables ← f.relatedVariables.erase(v)
 4: for i in f.combList do
 5:     i ← i.erase(v)
 6:     if i is in nF.combList then
 7:         nF.potentialsList[i] ← nF.potentialsList[i] + temp
 8:     else
 9:         nF.combList.append(i)
10:         nF.potentialsList[i] ← temp
11:     end if
12: end for
13: return nF
```

Algorithm 7 Construct Junction Tree from DAG

```
1: Moralization
2: Triangulation
3: Form a Junction Tree
```

Algorithm 8 Moraliazation

```
edgesM ← adjacency matrix initialized all zero
for nd in nodesContainer do
    i ← nd.index
    for pnd in nd.parents do
        j ← pnd.index
        edgesM[i][j] ← 1
        edgesM[j][i] ← 1
        for pnd2 in nd.parents and pnd2 ≠ pnd do
            k ← pnd.index
            edgesM[j][k] ← 1
            edgesM[k][j] ← 1
        end for
    end for
end for
```

The lines 6 and 7 are to record the existing edges.
The lines 10 and 11 are to ”marry” parents and record these edges.

Algorithm 9 Triangulation(G, Ord, Cliques)

Reference:
http://compbio.fmph.uniba.sk/vyuka/gm/old/2010-02/handouts/junction-tree.pdf
A chord of a cycle in a graph is edge connecting two vertices not adjacent in the cycle. A cycle is chordless if it has no chords. A graph is triangulated (also called chordal), if no cycle of length at least 4 is chordless.

Input: G: a graph; Ord: elimination ordering; Cliques: a set recording the cliques

```
if G.size = 1 then
    return
end if
v ← first node in Ord
vs ← v + all neighbours of v
c ← FormClique(vs)
Cliques.insert(c)
Triangulation(G-v, Ord-v, Cliques)
```

The line 1 can also using ”if Ord.size=1 then”. The line 6 ”FormClique(vs)” is to connect every pair of nodes in vs.

Algorithm 10 Form a Junction Tree (Cliques)

Input: Cliques: a set storing the cliques

```
for every pair of cliques c1 and c2 do
    sept ← separater(c1,c2)
    sept.weight ← number of common variables of c1 and c2
    connect(c1, sept, c2)
end for
JunctionTree ← maximum spanning tree by Kruskal's algorithm
return JunctionTree
```

Note that in line 1, pair c1 and c2 have no order, which means that (c1, c2) and (c2, c1) are the same pair. The line 4 is to form a link such that: (c1)—[sep]—(c2)

Algorithm 11 Assign and Initialize Potentials

```
1: for s in all separaters do
2:     Initialize every $\phi_s(x_s) \leftarrow 1$
3: end for
4: for c in all cliques do
5:     Initialize every $\phi_c(x_c) \leftarrow 1$
6: end for
7: factors $\leftarrow$ empty set to store factors
8: for n in nodes of the original Bayesian network do
9:     f $\leftarrow$ Construct a factor from n
10:    factors.insert(f)
11: end for
12: for f in factors do
13:    c $\leftarrow$ the minimum clique such that c's variables covers f's
14:    Multiply($\phi_c$, f)
15: end for
```

Maybe it is not necessary to find the *minimum* in the line 13. Maybe it can be any that covers. I am not sure yet.
We need to assure that each factor (a.k.a. potential) is asigned only once, which means that each potential should be assigned to only one clique. So that the product of the clique potentials is the unnomalized probability being poportional to the original joint probability.

Algorithm 12 Introduce Evidence to Junction Tree

Input: E: evidence (a.k.a. observations)

```
1: for c in all cliques do
2:     for x in all possible instantiations of c's variables do
3:         if conflict(x, E) then
4:             $\phi_c(x) \leftarrow 0$
5:         end if
6:     end for
7: end for
```

Algorithm 13 Update the whole Junction Tree with evidence

```
1: JT $\leftarrow$ construct a junction tree
2: introduce evidence to JT
3: r $\leftarrow$ select a root of JT
4: for child c of r do
5:     r.Collect()
6: end for
7: for child c of r do
8:     JT.Distribute()
9: end for
```

Algorithm 14 r.Collect()

```
1: for child c of r do
2:     c.Collect()
3:     pass message from c to r
4: end for
5: return  message of r to caller
```

Algorithm 15 r.Distribute()

```
1: for child c of r do
2:     pass message from r to c
3:     c.Distribute()
4: end for
```